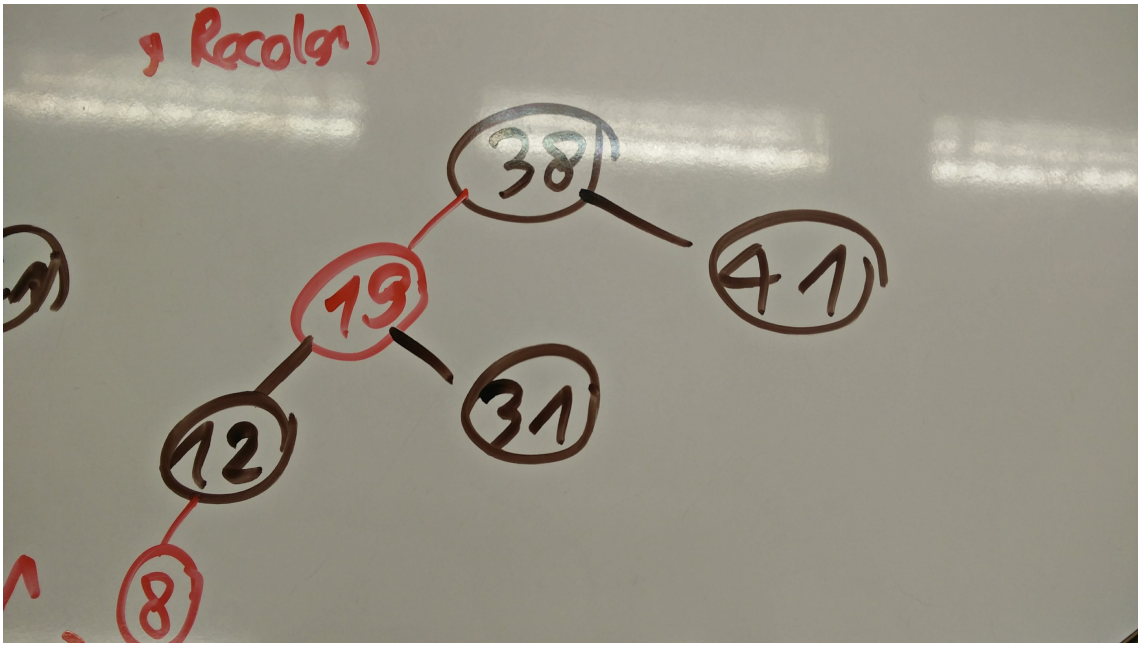
CS 271 - Project 0111

James Le

April 20, 2017

1. Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, and 8 into an initially empty red-black tree.



2. Prove that the longest path from a node x in a red-black tree to a descendant leaf has length at most twice that of the shortest path from x to a descendant leaf.

   Solution: From the red-black properties, we have that every simple path from node x to a descendant leaf has the same number of black nodes and that red nodes do not occur immediately next to each other on such paths. Then the shortest possible simple path from node x to a descendant leaf will have all black nodes, and the longest possible simple path will have alternating black and red nodes. Since the leaf must be black, there are at most the same number of red nodes as black nodes on the path.

3. Implement a red-black tree template class. Your implementation should follow the same guidelines as the binary search tree and include the same public methods.

4. Write another template class implementation of a Dictionary ADT that inherits from your red-black tree template class. Your class should include the same public methods as your Dictionary template classes in the previous projects/

5. Building on the comparison performed in last project, compare the time it takes to insert all of the movies with the red-black tree, hash table, and binary search tree implementations. Explain the results.

   Binary Search Tree: 11.529 seconds

   The binary search tree is significantly slower than either other data structure because the order of the movies in the movie file creates the worst case scenario tree. This occurs because the movies are in alphabetical order, which is effectively increasing order. Hence, each element inserted is greater than the element inserted before it, so inserting requires the function to

iterate to the right through the entire tree. As constructing the dictionary calls insert multiple times, the function ends up iterating through successively longer trees.

Hash Table: 0.144 seconds

Due to the effective hash function, each movie is assigned a very unique bucket, containing only a small number of additional movies. Since the buckets have almost the same number of movies, the time to go through each list within a bucket is almost the same in each bucket. Since each movie is inserted at the head of its respective bucket, inserting each movie is relatively fast, as the insertion method does not need to iterate to the end of the list.

Red-Black Tree: 1.188 seconds

The red-black tree adjusts itself when new movies are inserted in order to preserve the black height property. This means that iterating to any leaf from the root takes effectively the same amount of time. This self-balancing means that unlike the binary search tree, the red-black tree does not end up as the worst case scenario tree. The tree ends up much more spread out, so there are fewer items to iterate through with each insert. However, the self-balancing adds some extra time to the insertion, so the hash table is quicker.